Resumen teórico 2 concurrente

Teoría 6

Programa distribuido

Programa concurrente comunicado por mensajes. Supone la ejecución sobre una arquitectura de memoria distribuida, aunque puedan ejecutarse sobre una de memoria compartida (o híbrida).
Los procesos SOLO comparten canales (físicos o lógicos).

Mecanismo para procesamiento distribuido

- Pasaje de Mensajes Asincrónicos (PMA)
- Pasaje de Mensajes Sincrónico (PMS)
- Llamado a Procedimientos Remotos (RPC)
- Rendezvous

Relación entre mecanismos de sincronización

- Semáforos mejora respecto de busy waiting.
- Monitores combinan Exclusión Mutua implícita y señalización explícita.
- PM extiende semáforos con datos.
- RPC y rendezvous combina la interface procedural de monitores con PM implícito.

PMA (Pasaje de mensajes asincrónico)

Operación Send → un proceso agrega un mensaje al final de la cola (“ilimitada”) de un canal ejecutando un send, que no bloquea al emisor

Operación Receive → un proceso recibe un mensaje desde un canal con receive, que demora (“bloquea”) al receptor hasta que en el canal haya al menos un mensaje; luego toma el primero y lo almacena en variables locales.

Receive es una primitiva **bloqueante**, ya que produce un delay. Semántica: el proceso NO hace nada hasta recibir un mensaje en la cola correspondiente al canal. NO es necesario hacer polling.

Características de los canales

- ➢ Acceso a los contenidos de cada canal: atómico y respeta orden FIFO.

- ➢ En principio los canales son ilimitados, aunque las implementaciones reales tendrán un tamaño de buffer asignado.
- ➢ Los mensajes NO se pierden ni modifican
- ➢ empty(ch) → determina si la cola de un canal está vacía (usar con cuidado)

Los canales se pueden usar como:

- ➢ **MailBoxes:** Cualquier proceso puede enviar o recibir por alguno de los canales declarados
- ➢ **Input Port:** un canal tiene un solo receptor y muchos emisores
- ➢ **Link:** canal tiene un único emisor y un único receptor

Productores y consumidores (filtro)

Filtro: proceso que recibe mensajes de uno o más canales de entrada y envía mensajes a uno o más canales de salida. La salida de un filtro es función de su estado inicial y de los valores recibidos.
Esta función del filtro puede especificarse por un predicado que relacione los valores de los mensajes de salida con los de entrada.
Problema: ordenar una lista de N números de modo ascendente. Podemos pensar en un filtro Sort con un canal de entrada (N números desordenados) y un canal de salida (N números ordenados).

Clientes y servidores

Servidor: proceso que maneja pedidos (“requests”) de otros procesos clientes.
Cliente: envía un mensaje a un canal de requerimientos general, luego recibe el resultado desde un canal de respuesta propio

Monitores activos

La eficiencia de monitores o PM depende de la arquitectura física de soporte:

- Con MC conviene la invocación a procedimientos y la operación sobre variables condición.
- Con arquitecturas físicamente distribuidas tienden a ser más eficientes los mecanismos de PM.

Dualidad entre Monitores y Pasaje de Mensajes

Programas con Monitores		***Programas basados en PM***
• Variables permanentes	↔	Variables locales del servidor
• Identificadores de procedures	↔	Canal *request* y tipos de operación
• Llamado a procedure	↔	*send request( ); receive respuesta*
• Entry del monitor	↔	*receive request( )*
• Retorno del procedure	↔	*send respuesta( )*
• Sentencia *wait*	↔	Salvar pedido pendiente
• Sentencia *signal*	↔	Recuperar/ procesar pedido pendiente
• Cuerpos de los procedure	↔	Sentencias del "case" de acuerdo a la clase de operación.

Continuidad conversacional

Existen A alumnos que hacen consultas a D docentes. El alumno espera hasta que un docente lo atiende, y a partir de ahí le comienza a realizar las diferentes consultas hasta que no le queden dudas.
Los alumnos son los procesos "clientes", y los docentes los procesos "Servidores". Los procesos servidores son idénticos, y cualquiera de ellos que esté libre puede atender un requerimiento de un alumno.
Todos los alumnos pueden pedir atención por un canal global y recibirán respuesta de un docente dado por un canal propio

Pares (peers) interactuantes

NO ENTENDI NADA!!!!

Problema: cada proceso tiene un dato local V y los N procesos deben saber cuál es el menor y cuál el mayor de los valores.
Soluciones:

Simétrica: hay un canal entre cada par de procesos
Es la más corta y sencilla de programar, pero usa el mayor número de mensajes (si no hay broadcast). Pueden transmitirse en paralelo si la red soporta transmisiones concurrentes, pero el overhead de comunicación acota el speedup.

Centralizada y anillo usan n° lineal de mensajes, pero tienen distintos patrones de comunicación que llevan a distinta performance

- En centralizada, los mensajes al coordinador se envían casi al mismo tiempo sólo el primer receive del coordinador demora mucho.
- En anillo, todos los procesos son productores y consumidores. El último tiene que esperar a que todos los otros (uno por vez) reciban un mensaje, hacer poco cómputo, y

enviar su resultado. Los mensajes circulan 2 veces completas por el anillo Solución inherentemente lineal y lenta para este problema

Teoria 7

PMS (Pasaje de mensajes sincrónicos)

- ➢ Los canales son de tipo link o punto a punto (1 emisor y 1 receptor).
- ➢ El trasmisor queda esperando que el mensaje sea recibido por el receptor (el grado de concurrencia se reduce respecto de la sincronización por PMA)
- ➢ Las posibilidades de deadlock son mayores. El programador debe ser cuidadoso de que todas las sentencias send (sync_send) y receive hagan matching.

Ejemplo productor/consumidor

- Con PMS los pares send/receive se completarán asumiendo la demora del proceso que más tiempo consuma. Si la relación de tiempo fuera 10 a 1 significaría multiplicar por 10 los tiempos totales.
- Con PMA, al principio el productor es más rápido y sus mensajes se encolan. Luego el consumidor es más rápido y “descuenta” tiempo consumiendo la cola de mensajes.

Mayor concurrencia con PMA, para lograr el mismo efecto en PMS se debe interponer un proceso “buffer”

Ejemplo cliente/servidor

Cuando un cliente está liberando un recurso, no habría motivos para demorarlo hasta que el servidor reciba el mensaje, pero con PMS se tiene que demorar.

CSP-Lenguaje para PMS

- ➢ Canal: link directo entre dos procesos en lugar de mailbox global. Son halfduplex y nominados.
- ➢ Las sentencias de Entrada (? o query) y Salida (! o shriek o bang) son el único medio por el cual los procesos se comunican.
- ➢ Para que se produzca la comunicación, deben matchear, y luego se ejecutan simultáneamente.
- ➢ Efecto: sentencia de asignación distribuida.

Comunicación guardada

Las operaciones de comunicación (? y !) pueden ser guardadas, es decir hacer un AWAIT hasta que una condición sea verdadera → El do e if de CSP usan los comandos guardados de Dijkstra

Las sentencias de comunicación guardadas aparecen en ***if*** y ***do***.

if B_1; ***comunicación***$_1 \rightarrow S_1$;
• B_2; ***comunicación***$_2 \rightarrow S_2$;
fi

Ejecución:

1. Primero, se evalúan las guardas
    a. Si todas las guardas fallan, el if termina sin efecto.
    b. Si al menos una guarda tiene éxito, se elige una de ellas (no determinísticamente).
    c. Si algunas guardas se bloquean, se espera hasta que alguna de ellas tenga éxito.
2. Segundo, luego de elegir una guarda exitosa, se ejecuta la sentencia de comunicación de la guarda elegida.
3. Tercero, se ejecuta la sentencia Si .

Paradigma de interacción entre procesos

3 esquemas básicos de interacción entre procesos: productor/consumidor, cliente/servidor e interacción entre pares.
Estos esquemas básicos se pueden combinar de muchas maneras, dando lugar a otros paradigmas o modelos de interacción entre procesos.

Paradigma master/worker

El concepto de bag of tasks usando variables compartidas supone que un conjunto de workers comparten una “bolsa” con tareas independientes. Los workers sacan una tarea de la bolsa, la ejecutan, y posiblemente crean nuevas tareas que ponen en la bolsa
Analizaremos la implementación de este paradigma con mensajes en lugar de MC. Para esto un proceso manager implementará la “bolsa” manejando las tasks, comunicándose con los workers y detectando fin de tareas. Se trata de un esquema C/S.

Paradigma Heartbeat

Util para soluciones iterativas que se quieren paralelizar.

Usando un esquema “divide & conquer” se distribuye la carga (datos) entre los workers; cada uno es responsable de actualizar una parte.
Los nuevos valores dependen de los mantenidos por los workers o sus vecinos inmediatos.

Topología de red

Los procesadores están conectados por canales bidireccionales. Cada uno se comunica sólo con sus vecinos y conoce esos links.
Algoritmo Heartbeat: se expande enviando información; luego se contrae incorporando nueva información
Excesivo intercambio de mensajes los procesos cercanos al “centro” conocen la topología más pronto y no aprenden nada nuevo en los intercambios.

Resumen de chatgpt porque no entendía:
El algoritmo heartbeat es un mecanismo de detección de fallos que permite a los nodos en un sistema distribuido monitorear la actividad y estado de otros nodos. La idea principal es que cada nodo envía señales periódicas (latidos) a otros nodos para indicar que está en funcionamiento. Si un nodo deja de recibir un latido de otro nodo en un tiempo predefinido, asume que ese nodo ha fallado o está inactivo.

Ventajas y Limitaciones

- *Ventajas:*
    - *Permite detectar fallos de manera rápida y sencilla.*
    - *Facilita la implementación de sistemas de alta disponibilidad.*
- *Limitaciones:*
    - *La configuración de los tiempos de latido y de timeout es crítica: si son demasiado largos, la detección de fallos será lenta; si son demasiado cortos, pueden producirse falsos positivos.*
    - *Aumenta el tráfico de red, especialmente en sistemas con muchos nodos.*

En resumen, el algoritmo heartbeat es una herramienta útil para mantener la consistencia y disponibilidad de un sistema distribuido al permitir que los nodos monitoricen continuamente el estado de sus pares.

Paradigma Pipeline

Un pipeline es un arreglo lineal de procesos “filtro” que reciben datos de un puerto (canal) de entrada y entregan resultados por un canal de salida

Estos procesos (“workers”) pueden estar en procesadores que operan en paralelo, en un primer **esquema a lazo abierto** (W1 en el INPUT, Wn en el OUTPUT).

Un segundo **esquema es el pipeline circular,** donde Wn se conecta con W1 . Estos esquemas sirven en procesos iterativos o bien donde la aplicación no se resuelve en una pasada por el pipe.

En un tercer **esquema posible (cerrado)**, existe un proceso coordinador que maneja la “realimentación” entre Wn y W1

Cuadrito de chatgpt:

Esquema	Flujo de Datos	Uso Ideal	Ejemplo
Lazo Abierto	Lineal, de inicio a fin	Procesos sin retroalimentación	Procesamiento de imágenes
Lazo Circular	Cíclico, con retorno parcial	Procesos que requieren ajuste reiterativo	Procesamiento de audio
Lazo Cerrado	Continuo, con retroalimentación	Sistemas adaptativos o de control constante	Control de temperatura en un edificio

Paradigma Probe-Echo

Las arquitecturas distribuidas se pueden asimilar a los nodos de grafos y árboles, con canales de comunicación que los vinculan. DFS es uno de los paradigmas secuenciales clásicos para visitar todos los nodos en un árbol o grafo. Este paradigma es el análogo concurrente de DFS.

Prueba-eco se basa en el envío de un mensajes (“probe”) de un nodo al sucesor, y la espera posterior del mensaje de respuesta (“echo”). Los probes se envían en paralelo a todos los sucesores.

Los algoritmos de prueba-eco son particularmente interesantes cuando se trata de recorrer redes donde no hay (o no se conoce) un número fijo de nodos activos

Paradigma Broadcast

En la mayoría de las LAN cada procesador se conecta directamente con los otros. Estas redes normalmente soportan la primitiva broadcast:
Los mensajes broadcast de un proceso se encolan en los canales en el orden de envío, pero broadcast no es atómico y los mensajes enviados por procesos A y B podrían ser recibidos por otros en distinto orden.
Se puede usar broadcast para diseminar información o para resolver problemas de sincronización distribuida. Ejemplo: semáforos distribuidos, la base es un ordenamiento total de eventos de comunicación mediante el uso de relojes lógicos.
…..rari

Paradigma token passing

Al fin uno que se entiende!!!!
Un paradigma de interacción muy usado se basa en un tipo especial de mensaje (“token”) que puede usarse para otorgar un permiso (control) o recoger información global de la arquitectura distribuida. Un ejemplo del primer tipo de algoritmos es el caso de tener que controlar exclusión mutua distribuida.

Paradigma servidores replicados

Un server puede ser replicado cuando hay múltiples instancias de un recurso: cada server maneja una instancia. La replicación también puede usarse para darle a los clientes la sensación de un único recurso cuando en realidad hay varios.
Ejemplo: filosofos

- Modelo centralizado: los Filósofo se comunican con UN proceso Mozo que decide el acceso o no a los recursos.
- Modelo distribuido: supone 5 procesos Mozo, cada uno manejando un tenedor. Un Filósofo puede comunicarse con 2 Mozos (izquierdo y derecho), solicitando y devolviendo el recurso. Los Mozos NO se comunican entre ellos
- Modelo descentralizada: cada Filósofo ve un único Mozo. Los Mozos se comunican entre ellos (cada uno con sus 2 vecinos) para decidir el manejo del recurso asociado a “su” Filósofo

Teoria 8

RPC y Rendezvous

- El Pasaje de Mensajes se ajusta bien a problemas de filtros y pares que interactúan, ya que se plantea la comunicación unidireccional.
- Para resolver C/S la comunicación bidireccional obliga a especificar 2 tipos de canales (requerimientos y respuestas). Cada cliente necesita un canal de reply distinto
- Demoran al llamador hasta que la operación llamada se termine de ejecutar y se devuelvan los resultados

Diferencias

- Un enfoque es declarar un procedure para cada operación y **crear un nuevo proceso** (al menos conceptualmente) para manejar cada llamado (RPC porque el llamador y el cuerpo del procedure pueden estar en distintas máquinas). Para el cliente, durante la ejecución del servicio, es como si tuviera en su sitio el proceso remoto que lo sirve

- El segundo enfoque es hacer rendezvous con un proceso existente. Un rendezvous es servido por una sentencia de Entrada (o accept) que espera una invocación, la procesa y devuelve los resultados

RPC (Remote Procedure Call)

Los programas se descomponen en módulos (con procesos y procedures), que pueden residir en espacios de direcciones distintos.
Los procesos de un módulo pueden compartir variables y llamar a procedures de ese módulo.
Un proceso en un módulo puede comunicarse con procesos de otro módulo sólo invocando procedimientos exportados por éste.
Los procesos locales son llamados **background** para distinguirlos de las operaciones exportadas.
Un proceso (o procedure) en un módulo llama a un procedure en otro ejecutando:

call Mname.opname (argumentos)

Para un llamado local, el nombre del módulo se puede omitir

La implementación de un llamado intermodulo es distinta que para uno local, ya que los dos módulos pueden estar en distintos espacios: **un nuevo proceso sirve el llamado**, y los argumentos son pasados como mensajes entre el llamador y el proceso server.
Si el proceso llamador y el procedure están en el mismo espacio de direcciones, es posible evitar crear un nuevo proceso.
En general, un llamado será remoto se debe crear un proceso server o alocarlo de un pool preexistente.

Enfoques para proveer sincronización

- **Si ejecutan con Exclusión Mutua** las variables compartidas son protegidas automáticamente contra acceso concurrente, pero es necesario programar sincronización por condición.
- **Si pueden ejecutar concurrentemente** necesitamos mecanismos para programar exclusión mutua y sincronización por condición (cada módulo es un programa concurrente) podemos usar cualquier método ya descripto (semáforos, monitores, o incluso rendezvous).

Es más general asumir que los procesos pueden ejecutar concurrentemente (más eficiente en un multiprocesador de memoria compartida)

Hay unos ejemplos de c/s, time server, etc que no los puse aca porque no los pienso estudiar!!!

Rendezvous

RPC por si mismo sólo brinda un mecanismo de comunicación intermódulo. Rendezvous combina comunicación y sincronización

- Como con RPC, un proceso cliente invoca una operación por medio de un **call**, pero esta operación es servida por un proceso existente en lugar de por uno nuevo.
- Un proceso servidor usa una **sentencia de entrada** para esperar por un call y actuar.
- Las operaciones se atienden una por vez más que concurrentemente.

A diferencia de RPC el server es un proceso activo

ADA – Lenguaje con Rendezvous

Desde el punto de vista de la concurrencia, un programa Ada tiene tasks (tareas) que pueden ejecutar independientemente y contienen primitivas de sincronización.

Los puntos de invocación (entrada) a una tarea se denominan entrys y están especificados en la parte visible (header de la tarea).

Una tarea puede decidir si acepta la comunicación con otro proceso, mediante la primitiva accept

La tarea que declara un entry sirve llamados al entry con accept:

accept nombre (parámetros formales) do sentencias end nombre;

Demora la tarea hasta que haya una invocación, copia los parámetros reales en los parámetros formales, y ejecuta las sentencias. Cuando termina, los parámetros formales de salida son copiados a los parámetros reales. Luego ambos procesos continúan.